DASHing Towards Hollywood

Saba Ahsan
Aalto University
Espoo, Finland
saba.ahsan@aalto.fi

Stephen McQuistin
University of Glasgow
Glasgow, UK
sm@smcquistin.uk

Colin Perkins
University of Glasgow
Glasgow, UK
csp@csperkins.org

Jörg Ott
TU Munich
Munich, Germany
ott@in.tum.de

ABSTRACT

Adaptive streaming over HTTP has become the de-facto standard for video streaming over the Internet, partly due to its ease of deployment in a heavily ossified Internet. Though performant in most on-demand scenarios, it is bound by the semantics of TCP, with reliability prioritised over timeliness, even for live video where the reverse may be desired. In this paper, we present an implementation of MPEG-DASH over TCP Hollywood, a widely deployable TCP variant for latency sensitive applications. Out-of-order delivery in TCP Hollywood allows the client to measure, adapt and request the next video chunk even when the current one is only partially downloaded. Furthermore, the ability to skip frames, enabled by multi-streaming and out-of-order delivery, adds resilience against stalling for any delayed messages. We observed that in high latency and high loss networks, TCP Hollywood significantly lowers the possibility of stall events and also supports better quality downloads in comparison to standard TCP, with minimal changes to current adaptation algorithms.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Networks** → **Transport protocols**;

KEYWORDS

dynamic adaptive streaming over HTTP, DASH, multimedia streaming, head-of-line blocking, transport layer multistreaming

ACM Reference Format:
Saba Ahsan, Stephen McQuistin, Colin Perkins, and Jörg Ott. 2018. DASHing Towards Hollywood. In *MMSys'18: 9th ACM Multimedia Systems Conference, June 12–15, 2018, Amsterdam, Netherlands.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3204949.3204959

MMSys'18, June 12–15, 2018, Amsterdam, Netherlands

1 INTRODUCTION

Video has grown to be the dominant class of traffic on the Internet in recent years, and it is expected to grow further. This growth has been driven by a rising number of cord-cutters: users who have switched to consuming video solely over the Internet. HTTP Adaptive Streaming (HAS) protocols, including proprietary protocols such as Apple's HTTP Live Streaming (HLS) and the MPEG-DASH standard, underpin much of this traffic. HTTP uses TCP at the transport-layer, however, which is suboptimal for applications that wish to trade-off reliability and order for timeliness, including Internet video applications.

HAS protocols operate on a pull-based technique, driven by the client application. An HTTP server provides video in discrete chunks of equal duration. Each chunk is provided in several different encodings, each at a different bit-rate. The client requests each chunk in turn, determining the appropriate representation to request based on its rate adaptation algorithm. The client then plays out these chunks in order, using a buffer to attempt to reduce stalling behaviour when a chunk doesn't finish downloading in time to play. However, the application is bound by the reliable delivery semantics of TCP: the application *must* wait for undelivered chunks. Stalling for undelivered chunks affects quality-of-experience, not only with the stall itself, but with the signals that the delay provides to the rate adaptation algorithms. Transient network issues are amplified.

In this paper, we develop an MPEG-DASH system that uses TCP Hollywood [11, 12], a variant of TCP that provides an unordered, multi-streaming delivery model. The changes made in TCP Hollywood are intended to reduce the impact of losses on quality-of-experience in high-delay networks. In high-delay and high-loss networks, adopting it for HAS results in total stall duration that is seven times lower than that of standard TCP (i.e., TCP without the TCP Hollywood modifications). Further, we show small improvements in start-up delay and average media bit-rate.

Similar results could be achieved by using multiple simultaneous transport-layer connections. However, these connections maintain separate flow and congestion control states, resulting in degraded performance. Multi-streaming in HTTP/2 allows a single transport-layer connection to be used by multiple ordered streams. Sending each chunk within its own stream would remove application-layer head-of-line blocking, but not the head-of-line blocking introduced by TCP. Our approach allows for a single transport-layer connection to be used, with the benefits that come with this, while eliminating head-of-line blocking.

The remainder of this paper is structured as follows. Section 2 briefly introduces TCP Hollywood, and how its delivery semantics are useful for our application. Section 3 describes the changes required to our MPEG-DASH client and server to work across TCP Hollywood. The testbed environment is described in Section 4, and the evaluation results are discussed in Section 5. Finally, Section 6 discusses related work, and how our contribution fits with this, while Section 7 concludes.

2 TCP HOLLYWOOD

TCP Hollywood [11, 12] is an unordered and time-lined transport protocol. While maintaining wire compatibility with standard TCP, TCP Hollywood removes two sources of transport-layer latency: in-order delivery and reliability. In-order delivery results in head-of-line blocking: data is not made available to the application until all earlier data has been delivered. Providing reliability over a best-effort network requires retransmissions, adding latency while loss is detected and retransmissions are sent. TCP Hollywood uses message-oriented semantics, and eliminates head-of-line blocking by delivering messages to the receiver in the order that they arrive. TCP Hollywood also relaxes standard TCP's reliability guarantee, providing *partial* reliability instead. The TCP Hollywood API allows applications to specify a deadline for each message, after which it will no longer be sent – this means that if a retransmission of a lost message is unlikely to arrive before its deadline, it will not be sent; an unexpired message will be sent instead (i.e., the retransmitted TCP segment's payload is different to the original transmission). Making use of this functionality requires that the application be aware of the TCP Hollywood extensions and the content being sent, such that it can set a deadline for each message.

The scope of this work is to evaluate the impact of HAS-over-TCP Hollywood on video quality-of-experience, while minimising change at the application-layer (i.e., maintaining alignment with the MPEG-DASH standard). One of the core tenets of the MPEG-DASH architecture is that the application logic is driven by the client: it determines when, and at which rate, chunks are requested. This means that the server can be a generic HTTP server – much of MPEG-DASH's popularity is owed to this approach. Given that TCP Hollywood's deadline API requires state to be held at the server, we only consider the benefits of its unordered delivery feature in this paper.

The benefits of TCP Hollywood alone have been analysed [11, 12]: eliminating head-of-line blocking significantly reduces the latency in lossy networks. However, the impact of these savings is muted, unless the application is structured such that it can make use of them: data sent in each TCP Hollywood message must be independently (of other messages) useful. Under standard MPEG-DASH, when a frame is to be played out, but has not yet arrived, the application will stall waiting for the missing data to arrive. When using standard TCP, this is a sensible design choice: subsequent frames are not available to the application. However, under TCP Hollywood, the frames that arrive while the missing frame is retransmitted may be available to be played out: this makes skipping frames a viable design choice. Rather than waiting for the missing frame, it is better to use error concealment techniques to minimise the impact of loss (as far as possible) and continue play-back. However, rate adaptation algorithms have not been designed to consider loss, and must be reevaluated when TCP Hollywood is used.

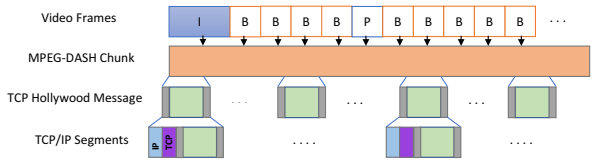

Figure 1: Illustration of the terminology used in this paper, and transport-layer encapsulation.

In introducing TCP Hollywood, and therefore out-of-order delivery, at the transport-layer, we must consider two design choices at the application-layer: the data that should be sent in each message, such that the benefits of out-of-order delivery are maximised, and the operation of the rate adaptation algorithm when some data may be missing. This paper will investigate these design choices.

3 UNORDERED DELIVERY IN MPEG-DASH

MPEG-DASH is designed for an ordered transport protocol. Adapting it for an unordered transport protocol requires several considerations. These changes are in two broad areas: the HTTP request and response semantics, and the rate adaptation algorithms. We consider both of these in turn below.

Given that our approach spans multiple layers of the stack, we must use carefully defined terminology. The MPEG-DASH standard sends *chunks*; these are groups of video *frames* that have the same duration. In our approach, the server will send these using TCP Hollywood *messages*. A chunk may be split across more than one message, but a message will never contain data from more than one chunk. Finally, messages are sent in TCP *segments*; a segment may contain data from more than one message, and may contain only some of the data for a message. Figure 1 illustrates this terminology.

3.1 HTTP Request/Response Semantics

As discussed in the previous section, the transmission unit of TCP Hollywood is a message: complete messages are delivered to the application when they arrive. Care must be taken in deciding what should be sent in each message: if a message is too large, then its loss will have a greater impact on the application's quality-of-experience; too small, and the ratio of payload to header will be low. Sending an entire chunk per message means that, in the event a segment that makes up the message is lost, upwards of 1 second of video data is not delivered to the application. Alternatively, the message boundaries can be aligned with the underlying digital container format. For example, MPEG-TS streams use 188 byte packets with Forward Error Correction (FEC). Sending each MPEG-TS packet as a message would minimise the impact of loss on QoE, but reduce the efficiency of our application by sending a TCP/IP header for every 188 bytes of payload. Sending a single video frame

is a reasonable choice for streams that don't allow for error recovery of a partially received frame, however, this comes at the cost of content-awareness at the server. In this study, we opt for a compromise: we send fixed-size (1400 byte) messages. This reduces the impact of loss (though not minimally), while amortising the header across a larger payload and preserving a content-agnostic HTTP server.

TCP Hollywood supports multi-streaming over a single TCP connection, allowing for the separation of the control and data channels. We use separate streams for HTTP request and response headers, sending each as separate TCP Hollywood messages. This allows the client to request later chunks while earlier chunks are still downloading. The server still sends chunks sequentially, but does not have to wait for the next request to arrive; this is especially useful in high latency networks. Figure 2 illustrates chunk retrieval with HTTP over both standard TCP and TCP Hollywood.

Our use of multi-streaming in TCP Hollywood is similar to HTTP/1.1 pipelining and bundle requests in HLS [13]. However, these approaches require quality estimates to be submitted for all chunks at request time. This means that a quality estimate may be two or more chunk durations old before being used at the server. Fluctuations in network conditions in the interval between request and response mean that these quality estimates can be incorrect. Under TCP Hollywood, requests can be sent while chunks are downloading, allowing for the server to receive a more recent and accurate estimate of network conditions. This reduces the likelihood of an under or over estimation of performance, improving quality-of-experience. Further, HTTP/1.1 pipelining suffers from transport-layer head-of-line blocking in standard TCP, which is eliminated under TCP Hollywood.

Finally, TCP Hollywood can discard late messages and continue delivering data to the application, thereby limiting stall events only to those cases when the play-out buffer is completely empty.

The requested chunks of video are transmitted on a single stream, using fixed size messages. As the server is content-agnostic, it is unable to set an expiry time for messages, and all messages are sent reliably. A content-aware server would set this to the relative presentation time of the chunk, subsequently reducing unnecessary retransmissions. For the sake of simplicity, the audio stream is not used. Each message is appended with a monotonically increasing sequence number and a stream offset. The stream offset is the offset of the last byte of the message within this stream, where a stream includes all chunks that have been transmitted previously. Both fields are used by the receiver to reorder the messages and detect losses. We expand the function of the play-out buffer to include de-jittering and reordering of the incoming messages.

The commonly used ISO Base Media File Format (MP4) is not suitable for TCP Hollywood due to its lack of tolerance for packet losses. Hence, we use MPEG Transport streams (MPEG-TS), which are allowed as part of the MPEG-DASH standard and also used in HLS.

3.2 Rate Adaptation Algorithms

Performance of an MPEG-DASH system relies heavily on a good rate adaptation algorithm. Such algorithms use application-level

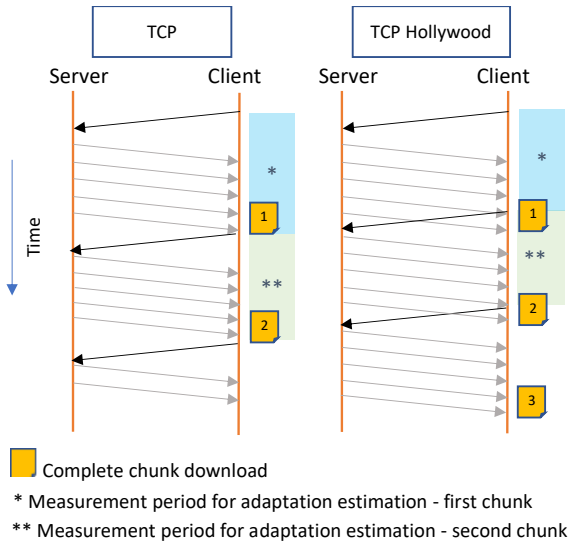

Figure 2: An illustration of chunk download with HTTP over TCP and over TCP Hollywood. The TCP Hollywood client requests the next chunk while the current chunk is still downloading. The shaded area represents the measurement period for each chunk.

measurements such as buffer length, throughput, time to download a chunk, or a combination of these to select an appropriate quality for download [1, 10, 14]. There is some consensus among researchers that buffer-based algorithms are more performant and reliable in a wider variety of network conditions, when compared with other algorithms [7, 9].

Our TCP Hollywood-based MPEG-DASH client uses the open-source BOLA (Buffer Occupancy based Lyapunov Algorithm) rate adaptation algorithm [14] adapted from the MPEG-DASH reference client implementation, dash.js.[1] BOLA uses the amount of video currently buffered to calculate the quality level (bit-rate level) for the next chunk. For live video, where buffer sizes are limited, the algorithm's implementation uses throughput estimates to further reduce over-estimation and stalling. Algorithm 1 shows the operation of rate adaptation using BOLA in a typical TCP based DASH client. Note that the quality estimation logic of the BOLA algorithm was adapted from dash.js without any modifications and remains unchanged (even when used in our TCP Hollywood-based approach), hence we do not further explain the inner workings of the estimation process.

In the presence of loss and duplication, as is possible with TCP Hollywood, the calculation mechanisms for measurements have to be modified. BOLA algorithm is written for standard TCP, where head-of-line blocking means that the client must wait for all data to arrive. In TCP Hollywood, head-of-line blocking is eliminated, allowing the client to continue even if some data hasn't arrived. Therefore, we introduce a new parameter, Rx_T, the Receive Threshold: when the ratio of the bytes received in a chunk reaches Rx_T, the

[1] https://github.com/Dash-Industry-Forum/dash.js/wiki

Algorithm 1 BOLA rate adaptation under TCP

```
1:  procedure DownloadStream
2:      n ← index of current chunk
3:      N ← Total number of chunks
4:      B ← Maximum Buffer Length
5:      b_now ← Current Buffer Length
6:      S_n^q ← Size of chunk with index n and quality q
7:      B_n^q ← Bytes received for chunk with index n and quality q
8:      while n < N do
9:          if b_now > B then
10:             Δt ← B − b_now
11:             WaitForDuration(Δt)
12:         end if
13:         q ← GetBolaQualityEstimate(b_now)
14:         DownloadChunk(q, n)
15:     end while
16: end procedure
```

client continues with its rate adaptation as if the entire chunk had arrived. Care is needed when selecting the value of Rx_T,: if it is too low, the rate adaptation will be performed on a partial view of network conditions, which may result in selecting an unsuitable rate; setting Rx_T too high would diminish the impact of eliminating head-of-line blocking. After experimenting with a range of Rx_T values between 0.75 and 0.99, under stable network conditions, we chose 0.9. We found this value was sufficiently high to include the current chunk as part of the buffered duration (only 10% of the chunk remains to be downloaded, with some data in-flight, and a limited number of messages that may be delayed and subsequently discarded by TCP Hollywood if they miss their deadline) and gather measurements for the next chunk, while sufficiently low as to enable the benefits of TCP Hollywood. Our experiments with Rx_T only considered 1 second chunks, however, we expect this value to hold for other chunk durations. Finally, with TCP Hollywood, some messages may be discarded due to late arrival. In order to prevent over-estimation in the presence of losses, we reduce the length of the buffer level by one chunk duration in this case. If the buffer level is under one chunk duration, the algorithm would any way select a low bit rate. In the current model, the algorithm does not differentiate between losses and would behave the same regardless of the type of loss. We discuss this further in Section 5.5 as an avenue for future work. The modified operation for TCP Hollywood is shown as Algorithm 2. Note that throughput measurements used by the BOLA algorithm, while not shown as part of the pseudocode for the sake of clarity, will also be calculated using only the partial download of the chunk.

The Rx_T parameter can be used with other adaptation algorithms as it impacts only the measurements and not the quality estimation mechanism. Preliminary testing for this study was carried out using both the buffer-based BOLA algorithm and the throughput-based Probe And Adapt (PANDA) algorithm [10]. Generally, we found BOLA outperformed PANDA for standard TCP and TCP Hollywood, in terms of stall avoidance and bit rate switching, a finding confirmed by previous studies [9]. Note that the main advantages

Algorithm 2 BOLA rate adaptation under TCP Hollywood

```
1:  procedure DownloadStream
2:      n ← index of current chunk
3:      N ← Total number of chunks
4:      B ← Maximum Buffer Length
5:      b_now ← Current Buffer Length
6:      S_n^q ← Size of chunk with index n and quality q
7:      B_n^q ← Bytes received for chunk with index n and quality q
8:      Rx_T ← Receive Threshold
9:      L ← Losses since last download
10:     T ← Duration of chunk
11:     while n < N do
12:         if b_now > B then
13:             Δt ← B − b_now
14:             WaitForDuration(Δt)
15:         end if
16:         if L > 0 & b_now > T then
17:             b_now ← b_now − T
18:         end if
19:         q ← GetBolaQualityEstimate(b_now)
20:         InitiateChunkDownload(q, n)
21:         WaitUntil(B_n^q < Rx_T * S_n^q)
22:     end while
23: end procedure
```

of TCP Hollywood (shown in Figure 2) are the removal of head-of-line blocking, and the addition of chunk request pipelining. These remain regardless of the algorithm used.

4 EVALUATION METHODOLOGY

We evaluate MPEG-DASH performance using both standard TCP and TCP Hollywood, to understand how the changes to the transport protocol affect quality of experience for streaming video. Both standard TCP and TCP Hollywood use the CUBIC congestion control algorithm; both would be impacted by any change to the variant of TCP used.

Our evaluation setup consists of a virtual environment that uses Mininet[2] to create a virtual network with a modified Linux kernel including the TCP Hollywood patches. We used our own MPEG-DASH server and client implementation. The TCP Hollywood API was disabled when testing with standard TCP, and a persistent HTTP/1.1 connection was used in both cases. The virtual network used included a single client and server connected through a switch. Path characteristics were emulated at the server interface using netem[3] Token Bucket Filter traffic control, with a fixed 5Mbps line rate, a reasonable value for our test videos, where the highest quality is 5600kbps. The test cases are divided into two categories: (i) a loss rate of 0.2% (random loss) with network latencies (RTT) between 150 to 400ms; and (ii) loss rates between 0 and 1% with network latency of 100ms. Additional testing was carried out with varying network conditions to evaluate adaptability.

[2] https://mininet.org
[3] https://wiki.linuxfoundation.org/networking/netem

Index	Encoding bit-rate (kbps)	Chunk bit-rate (kbps)	PSNR	SSIM
1	1200	1301	37.220176	0.955177
2	1800	1941	39.142917	0.967720
3	2400	2584	40.478799	0.974580
4	3000	3228	41.515695	0.978962
5	3500	3766	42.226845	0.981527
6	4000	4304	42.845958	0.983509
7	5000	5328	43.868625	0.986314
8	5600	6030	44.387282	0.987527

Table 1: Video representations; all representations have a resolution of 1920x1080p. PSNR and SSIM of re-encodings calculated against original Y4M reference.

We used Big Buck Bunny[4] with 8 quality levels as a video test sequence. The encoding bit-rates ranged from 1200 to 5600kbps with a constant resolution of 1920x1080p. The constant resolution eliminates the need for re-scaling in objective Quality of Experience (QoE) computations, which require the videos to be of the same resolution. MPEG-TS is used to allow loss recovery. The use of MPEG-TS adds about 10% in metadata overhead to the streams in our case (the use of MPEG-TS is not essential to our approach, and any encoding or container format that is robust to packet loss could be used). The encoding details for the different quality levels of the test sequence are given in Table 1. As TCP Hollywood is designed for low-latency applications, we use a 16 second buffer, with a 1 second chunk duration.

To compare the performance of MPEG-DASH over standard TCP and TCP Hollywood, we use a number of Quality of Service (QoS) and QoE metrics:

Start-up delay
The amount of time from the start of the test until 2 seconds of video has been downloaded and demuxed.

Stall duration
The total amount of time the video stalls due to a completely empty buffer. The client would wait for one additional chunk to be downloaded before resuming play-out when a stall occurs.

Adaptability and stability
Adaptability is a measure of how quickly the adaptation algorithm can adapt to change in network conditions and stability is characterised by the client's ability to mitigate frequent bit rate fluctuations. These are measured using two metrics: (i) the average bit-rate of the downloaded chunks (adaptability), and (ii) the percentage of chunks with a downward bit-rate switch during play-out (stability).

Objective QoE
To measure the level of distortion produced due to a discarded message, we use the objective QoE metrics Peak Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM). Both metrics are full-reference. We use the original Y4M sequence as reference and FFmpeg[5] for the computation. Since PSNR and SSIM values will be lower for lower quality chunks, the metrics also provide a measure of quality even in the absence of discarded messages.

These metrics were chosen since they are widely used and have been shown to be representative of different aspects of the user experience for MPEG-DASH video.

The code used in our evaluations is described in Appendix A.

5 RESULTS

In this section, we present the findings of our evaluation. All test cases were repeated ten times and the depicted values are trimmed averages taken after discarding the highest and lowest values. Furthermore, graphs contain error bars, which extend from the 20^{th} to 80^{th} percentile unless explicitly indicated otherwise. Note that due to the random nature of losses in the network emulated during testing, outliers are inevitable, which is why we have chosen to focus on the middle population of the values. To the best of our knowledge, this approach does not produce any bias towards either protocol.

5.1 Stall Events

The single most impactful impairment for MPEG-DASH video user experience has been found to be stall events [6]. A perfect adaptation algorithm would eliminate stall events during video play-out by pre-buffering content and by adjusting the download quality of the video to match the network performance. However, in reality, stall events occur, due to imperfect estimates of network conditions.

As we might expect, TCP Hollywood offers most benefit in the presence of moderate losses and high delay. Figure 3 shows the effect of network latency on the total stall duration experienced over standard TCP and TCP Hollywood when the line rate and loss rate were kept constant at 5Mbps and 0.2% respectively. At lower network latency, loss recovery is quick and hence TCP Hollywood and standard TCP have similar performance. However, for higher network latency, head-of-line blocking in standard TCP prevents data that has already arrived from reaching the application. When the missing TCP segment does arrive, the client can read a larger amount of data into its play-out buffer. On average, no more than one or two TCP Hollywood messages are discarded at network latencies of 300ms and 350ms. The additional performance comes from the ability of the rate adaptation algorithm in our TCP Hollywood-based MPEG-DASH client to compute metrics and decide the quality of the next chunk before the delivery of the current chunk (see Section 3). This would mean that the client does not have to wait unnecessarily for the arrival of retransmissions before requesting the next chunk, helping to avoid stalls.

When loss is high and delay is low, MPEG-DASH over TCP Hollywood and over standard TCP generally have similar performance. As packet loss increases, the TCP Hollywood-based client sees some performance benefit. Figure 4 shows the results when network latency is 100ms, line rate is 5Mbps and different network loss rates are used. When using loss rates of 0.6% and 0.8%, we observed that the TCP Hollywood-based client is able to avoid some stall events by discarding some late arriving messages, however the benefit is very small in comparison to standard TCP as loss recovery is fast at low RTTs. At loss rates of 1%, we observe that

[4]https://peach.blender.org
[5]https://ffmpeg.org/ffmpeg-filters.html

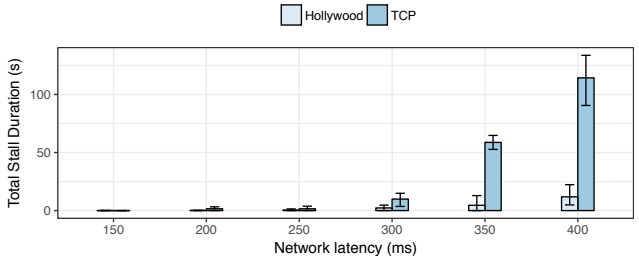

Figure 3: Stall events. Test networks were 5Mbps with 0.2% loss with variable delay. The client using standard TCP suffers more from stalls than the client using TCP Hollywood, with the latter being able to minimise stalls in the presence of high network delay.

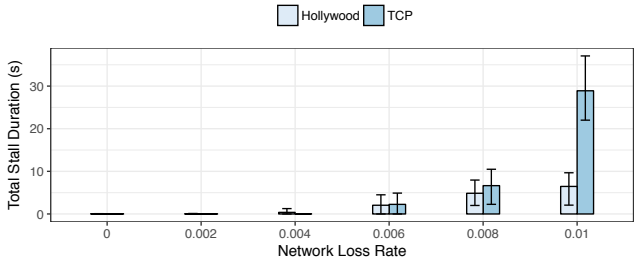

Figure 4: Stall events. Test networks were 5Mbps with 100ms delay and variable loss rates. With high loss rates, loss recovery can be delayed due to lost retransmissions, leading to more stall events for client using standard TCP.

the advantage of using TCP Hollywood is significant, keeping the stall values for that client at around 5 seconds, in comparison with around 30 seconds in the case of the client using standard TCP for a 10 minute video. At high loss rates, the possibility of losing a retransmitted TCP segment become higher and loss recovery becomes a time consuming process even at lower network delays.

5.2 Start-up Delay

We measure the start-up delay from the beginning of the test to the point when two seconds of video (two chunks) have been downloaded and buffered, including the time taken to fetch the manifest file. We use persistent connections for both standard TCP and TCP Hollywood, allowing the TCP connection establishment and slow start latencies to be amortised across a longer duration. In the case of TCP Hollywood, play-out can begin before the download of the second chunk is complete, unlike standard TCP. However, as we use a receive ratio of 0.9, the advantage is not significant for high speed networks. We still observe faster startup values for TCP Hollywood in all scenarios because of its ability to request chunks earlier. The higher the network latency, the greater the benefit of TCP Hollywood.

Figures 5 and 6 show startup delay values observed in our experiments. In the presence of higher losses, the startup delay can become less stable and statistical significance in average values is

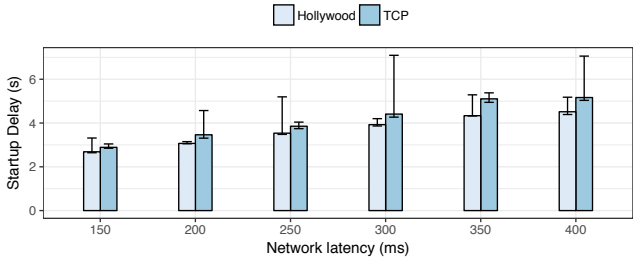

Figure 5: Start-up delay. Test networks were 5Mbps with 0.2% loss with variable delay. The higher the network latency, the greater the benefit of TCP Hollywood.

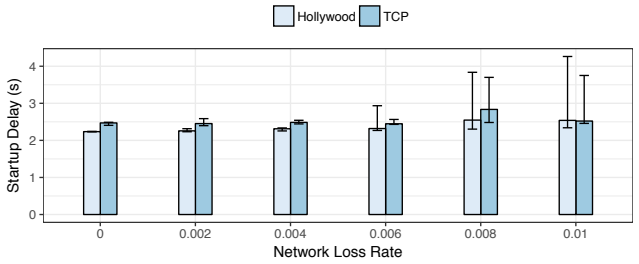

Figure 6: Start-up delay. Test networks were 5Mbps with 100ms with variable loss rates. The start-up delay is slightly lower for the TCP Hollywood-based client, however, randomness of emulated losses makes the values unpredictable.

lowered. For this reason, the median value after eliminating the highest and the lowest observed values are shown in the graphs.

5.3 Adaptation and Stability

While stall durations can be significantly reduced for high delay and high loss networks by eliminating head-of-line blocking using TCP Hollywood, these improvements only hold real significance if a TCP Hollywood-based client is able to deliver them while maintaining the same level of video quality as a standard TCP-based client. Figure 7 shows the effect of network latency on the average bit-rate of the video chunks for each transport protocol. The error bars indicate the standard deviation in the downloaded chunk bit-rate; a higher standard deviation indicates a higher range of bit-rate switching. It can be seen that TCP Hollywood has a higher average bit-rate with a similar level of standard deviation when network delay is 150ms and 200ms. For higher network delay, TCP Hollywood continues to deliver chunks at a similar average bit-rate value than standard TCP. In general, standard TCP appears to be more stable, as exhibited by the percentage of chunks with a rate drop shown in Figure 8. A rate drop is counted each time a chunk is downloaded at a bit-rate which is lower than the chunk immediately before it. The difference in rate drops is equivalent to about 1 additional drop observed per minute for TCP Hollywood than for standard TCP. We show only the rate drops here as they have a higher impact on user experience than upward switching, however, the observed trends were similar for both.

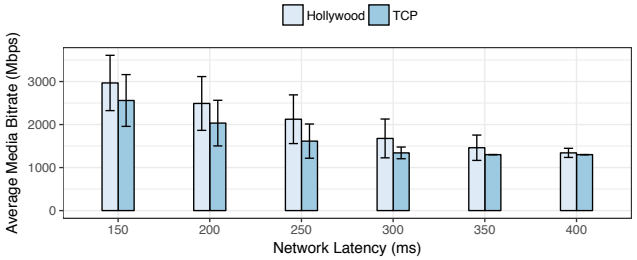

Figure 7: Adaptation and stability. Test networks were 5Mbps with 0.2% loss with variable delay. The error bars represent standard deviation, indicating bit-rate switching. For higher delay values, standard TCP shows lower standard deviation because it remains mostly at the lowest available bit-rate.

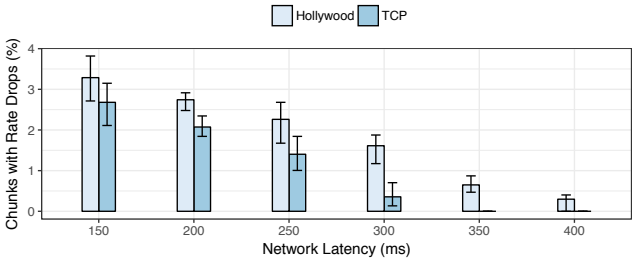

Figure 8: Adaptation and stability. Test networks were 5Mbps with 0.2% loss with variable delay. The graph shows the percentage of chunks that were downloaded at a lower rate than the previous one. The standard TCP adaptation algorithm is more stable for all networks. The test video had 597 chunks.

We observe that when packet loss is kept under 0.1%, clients using both standard TCP and TCP Hollywood exhibit similar behaviour. As shown in Figure 9, where network delay is 100ms, the TCP Hollywood-based client was able to deliver a higher average bit-rate than the standard TCP-based client, as loss rates increased. Figure 10 shows the stability of the rate adaptation algorithm under both standard TCP and TCP Hollywood. At a loss rate of 0.2%, we found that the TCP Hollywood-based client had higher stability than the standard TCP-based client. However, for higher loss rate values, the TCP Hollywood-based client is less stable, as it attempts to download higher bit-rate chunks, while the standard TCP-based client maintains stability at the lowest bit-rate.

5.4 Quality of Experience

None of the previously discussed metrics quantify the impact of the dropped messages on a TCP Hollywood-based DASH client. A discarded message will result in visual impairments that are not possible for a client using a reliable TCP stream. Figure 11 and 12 show the observed PSNR and SSIM values of the received streams for the different transport options. The figures show box plots of frame-level PSNR and SSIM values observed during multiple

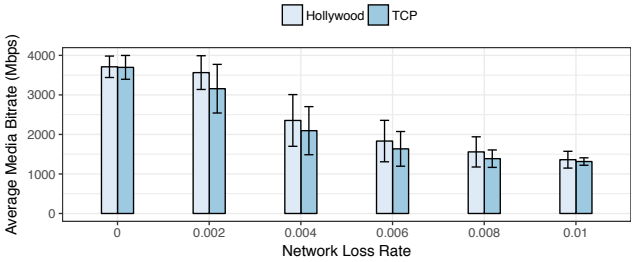

Figure 9: Test networks were 5Mbps with 100ms with variable loss rates. TCP Hollywood significantly outperforms standard TCP at loss rate of 0.2% with a higher average rate and lower standard deviation. Error bars represent standard deviation.

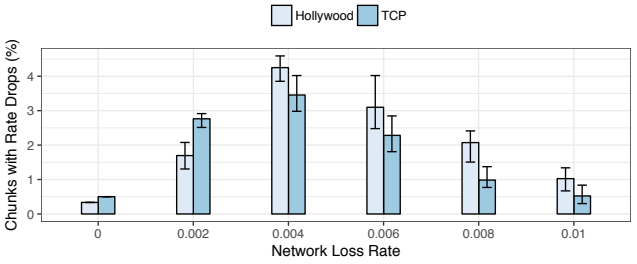

Figure 10: Test networks were 5Mbps with 100ms with variable loss rates. Hollywood is more stable for 0.2% and no loss cases, however, at higher loss levels, standard TCP becomes more stable.

iterations of the test. Note that although the two objective QoE metrics represent the quality of the chunks and the level of visual impairments due to discarded packets, it does not show the impact of stall events, which were discussed in Section 5.1. Across all of the tests we ran in these experiments, about 30% of streams discarded at least one message during the 10 minutes of the video, however, of these only 5% lost more than 10 messages. Furthermore, the impact of a lost message is most evident when a part of an I frame is lost. However, since chunks begin with an I frame, the effect of even the worst visual impairment does not propagate farther than the duration of the chunk, which in our case is 1 second.

5.5 Other Testing

We represent in this paper results from cases where the stall events were within acceptable limits for a range of latency and loss values. However, we tested for other conditions as well. For delay variation testing with a fixed loss rate at 0.03%, we found that both protocols performed equally with no stall event for delays as high as 400ms. For a loss rate of 0.8%, the stall duration was about 300s even at a delay of 200ms for the standard TCP-based client. For the TCP Hollywood-based client, the stall events were reduced by 40% but nearly 150 messages were discarded. Similarly, for a loss rate of 2%, stall durations were as high as 300s when running over standard TCP with a 100ms network latency, while the use of TCP

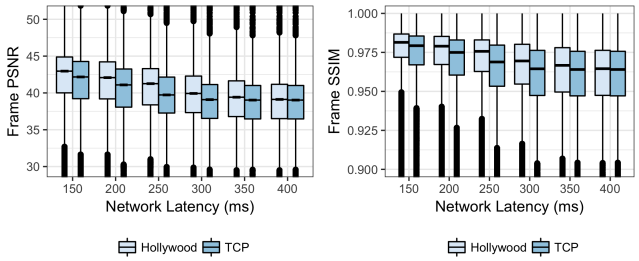

Figure 11: Impact of delay on quality of experience. Test networks were 5Mbps with 0.2%loss with variable delay.

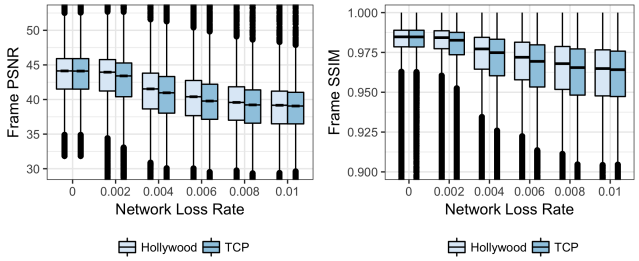

Figure 12: Impact of packet loss on quality of experience. Test networks were 5Mbps with 100ms with variable loss rates. The TCP Hollywood-based client maintains a higher PSNR and SSIM than standard TCP.

Hollywood reducing stall durations to around 230s at the cost of heavy losses. Given that such high levels of stalling will any way be unacceptable to users, we feel that attempting to evaluate which protocol performed better is of little consequence.

To evaluate the suitability of the adaptation algorithm in fluctuating network conditions, we ran tests using the twelve network profiles (*a* - *l*) used by Spiteri et. al. in their evaluation of the BOLA algorithm [14]. The profiles cover a wide range of loss rates, delays and bandwidths and network conditions are changed during the tests every 30 seconds. Since some bandwidths in the test were lower than the lowest bit-rate in our test videos and the buffer length was limited to 16 seconds, we observed stall events, especially for the more aggressively changing profiles (profiles *g* to *l*). Generally, the use of TCP Hollywood lowered the stall events, as shown in Figure 13. Start-up delay was similar for clients using both protocols, although some profiles with longer network delays in the beginning benefited slightly from the use of TCP Hollywood as shown in Figure 14.

A violin plot of the PSNR and SSIM values shows the frequency of each value in Figure 15. The similarity in the shape of the plot for clients using both standard TCP and TCP Hollywood shows that both protocols downloaded similar quality chunks, as observed in the average bit-rate results shown in Figure 16. The TCP Hollywood-based client had a lower number of downward rate switches than the standard TCP-based client for almost all profiles, as shown in Figure 17. The client using BOLA working with TCP Hollywood generally adapted faster to a higher bit-rate and was able to sustain

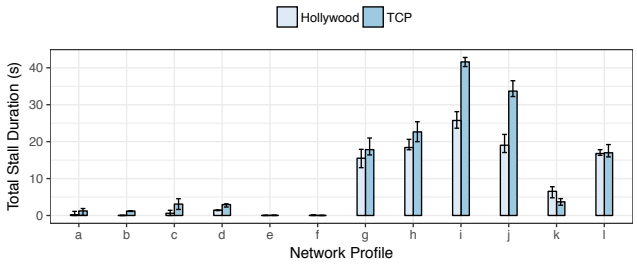

Figure 13: Stall events are reduced for TCP Hollywood for all except profile k, where latencies are under 20 ms and bandwidth levels between 9Mbps and 1Mbps.

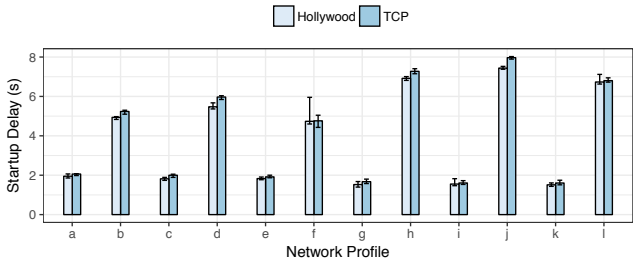

Figure 14: Startup delays observed for different network profiles.

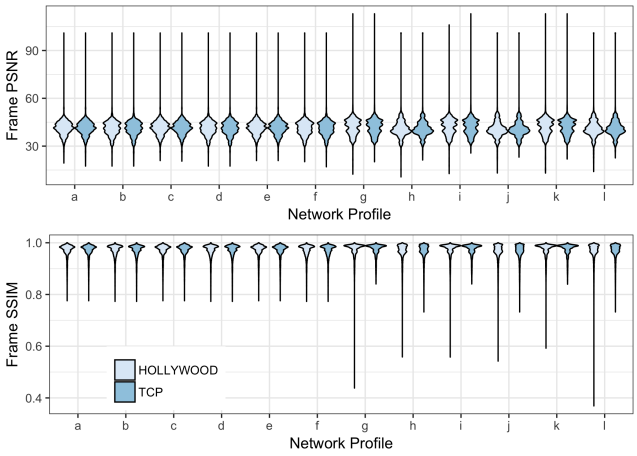

Figure 15: Quality of experience profiles. PSNR and SSIM values of various network profiles revealed similar results for both standard TCP and TCP Hollywood-based clients. The shape of the violin graph reveals more information about the frequency of values, showing the similarity in the download quality of both protocols with the tails in SSIM plot representing the effect of lost messages.

a higher bit-rate for longer than the client using standard TCP. However, this also meant that the TCP Hollywood client would drop suddenly to a lower rate while the standard TCP-based client would

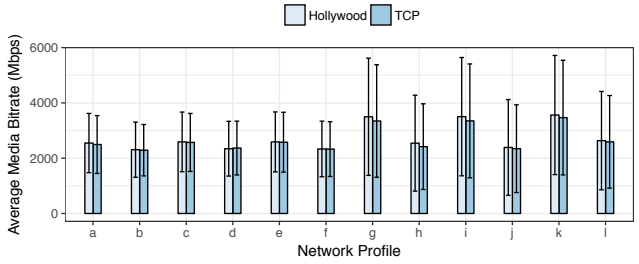

Figure 16: The average media bit rate was similar for all network profiles. The slight improvement for TCP Hollywood is further explained in Figure 18.

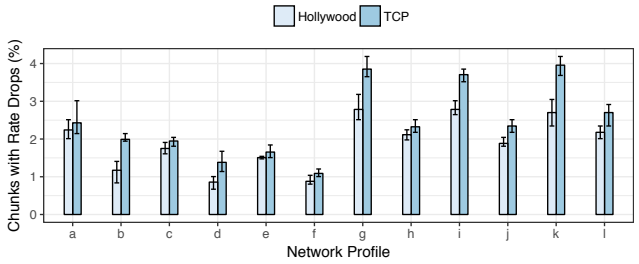

Figure 17: Fewer rate drops were observed over TCP Hollywood for all network profiles.

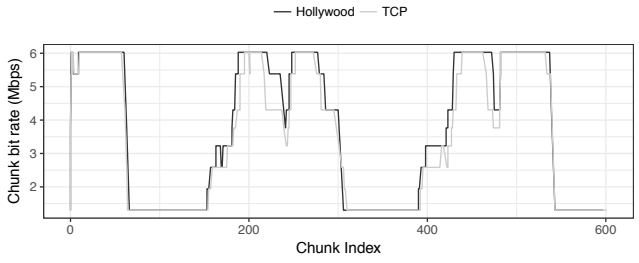

Figure 18: TCP Hollywood allows the adaptation algorithm to adapt more quickly to higher bit-rates and also to sustain them for longer periods in the presence of network degradation.

drop gradually. If the degradation lasted less than the 30 seconds defined by the profiles, as may be the case for real networks, the TCP Hollywood-based client is more likely to sustain the current bit-rate without dropping at all. A sample bit-rate adaptation for network profile *g* is shown in Figure 18. Note from Figure 15 that the TCP Hollywood-based client does suffer from message losses in profile *g* and future work should include exploring improvements in the adaptation algorithm to limit the number of discarded messages in favour of stalls in the presence of high losses and possibly detecting and adapting differently for different kinds of loss.

6 RELATED WORK

HTTP adaptive streaming has become the de-facto standard for on-demand video streaming. MPEG-DASH, an International Standard, is similar to other flavours of HAS, in that the architecture involves a client requesting discrete chunks of media over HTTP, at a rate determined by its adaptation algorithm. HAS protocols, by virtue of their use of chunked encodings and HTTP over TCP, introduce significant latency, and rely heavily upon buffering to maintain quality-of-experience. However, a previous study has shown that latency, in a local network, can be reduced to as low as 240ms[4]. The cost of this was higher packaging overhead, with increases of up to 13%.

An obvious application-layer approach to reducing latency in MPEG-DASH is to reduce the size of chunks; however, doing this leads to a significant increase in the number of HTTP requests sent. Wei and Swaminathan [16] propose using HTTP/2's server push mechanism to reduce the number of requests, potentially only requiring a single HTTP request across the entire stream. Xiao et al. [17] take a similar approach, but focus on mobile devices. While HTTP/2 provides server push and multi-streaming (and so eliminates head-of-line blocking at the application-layer), using standard TCP means that the application will still be impacted by head-of-line blocking. In the event of loss, data will be delayed by more than one RTT [12]; this is problematic in low-latency applications. The use of multiple simultaneous TCP connections provides a delivery model that is analogous to a multi-streaming protocol. However, these connections do not share flow and congestion control state, which degrades their performance. Further, managing these connections introduces complexity at the application-layer.

Our approach attempts to eliminate the latency associated with head-of-line blocking by using TCP Hollywood, whose message-oriented unordered delivery model is well suited to low-latency applications. Other transport-layer solutions include QUIC [8], a UDP-based protocol with support for stream multiplexing. Evaluations conducted over QUIC [2], without application-layer changes, have shown that MPEG-DASH QoE is degraded.

Neither of these approaches – application-layer changes (e.g., using HTTP/2) or novel transport-layer protocols (e.g., TCP Hollywood or QUIC) – alone is sufficient to improve application performance. Both are required: application-layer changes are necessary, but these must be supported by the semantics of the underlying transport protocol. The application-layer modifications we propose here are likely to be compatible with other multi-streaming transport-layer protocols, including QUIC.

Complementary latency-reducing approaches include a server and network-assisted variant of MPEG-DASH, SAND [15]; this is currently in development, and will form part of the MPEG-DASH standard. SAND enhances MPEG-DASH with asynchronous network-to-network and network-to-client quality-related message exchange. A Software Defined Networking (SDN) based approach has been shown to improve user QoE by providing the client with network performance and cache content information to assist the cache and bit-rate selection while using the SDN network to optimise the caches [3]. Additionally, Frömmgen et al. [5] propose a programming model for multipath TCP scheduling; the delivery model of TCP Hollywood, combined with the type of traffic being

carried, make this an interesting basis for future work. For example, exposing message deadlines to a multipath scheduler may increase the proportion of messages that meet their deadline.

7 CONCLUSION

In this work, we evaluated the effects of unordered delivery on MPEG-DASH clients, using TCP Hollywood. We observed that by eliminating head-of-line blocking on a chunk and message level, an MPEG-DASH client using TCP Hollywood is able to significantly reduce stall events while also slightly reducing start-up delay and improving download quality. The observations and lessons learnt during the course of this study can also be applied to other transports that support non-reliable streams, such as SCTP and QUIC.

A REPRODUCIBILITY

To aid with reproducibility, we provide all of the source code used in generating the results described in this paper, alongside a Makefile that describes and performs the process of performing the experiments, processing and graphing the results, and producing the paper. The source code for the paper is available at http://dx.doi.org/10.5525/gla.researchdata.596.

Our evaluations require a modified Linux kernel, and each run of the evaluation simulates different network parameters. To manage this complexity, we have split the paper's build process into a series of stages. In this section, we describe the inputs and outputs of each stage; we also give approximate durations for each stage. We start by describing the required dependencies.

Dependencies

The experiments are conducted within virtual machines, that use Linux with the TCP Hollywood kernel extensions and API installed. VirtualBox and Vagrant are used to manage these virtual machines. Each experiment run involves streaming Big Buck Bunny over a simulated network; after this, FFmpeg is used to perform SSIM and PSNR analysis between the streamed version and a reference copy. Finally, Python and R are used to analyse and graph the results of the experiments. The versions that were used in our testing are shown in brackets; other versions may work.

- FFmpeg (3.4.2)
- Python (2.7)
- R (3.4.3) and packages (rkvo, ggplot2, data.table)
- Vagrant (2.0.2)
- VirtualBox (5.2.6r120293)
- xz (5.2.3)

The experiments use the TCP Hollywood kernel and API. These are located in the following repositories, and the versions used by the experiments are specified in the Makefile:

- Kernel: https://github.com/lumisota/tcp-hollywood-linux
- API layer: https://github.com/lumisota/hollywood-api-video

These versions of the TCP Hollywood kernel and API included with the source code for the paper, in the data repository version described above.

For readability, whenever this section refers to TCP Hollywood, it is these versions that have been used.

Mininet[6] (version 2.2.1) is used to simulate the network for each run. This is installed, and run, within the TCP Hollywood Vagrant box, and is not required to be installed on the host machine.

Stage 0 (15 minutes)

Where the TCP Hollywood kernel source code has not been provided, the Makefile will pull a copy of it, using the repository and version specified. This takes place within an Ubuntu 14.04 virtual machine, which clones the repository, and compresses it into a tarball; this tarball is used in the next stage.

By performing this stage inside a Linux virtual machine, rather than on the host machine, we avoid problems caused by case-insensitive file systems. The Linux kernel assumes case-sensitivity, and there are a number of files in the same directory whose name only differs in case. These files would be deleted if the repository was cloned on a host that had a case-insensitive file system; macOS, by default, uses such a file system.

The data repository package contains the TCP Hollywood source code, as a tarball, in the stage0 directory.

Stage 1 (3-4 hours)

TCP Hollywood is comprised of a set of modifications to the Linux kernel, and a user-space API layer. Stage 1 involves building a virtual machine image that has the TCP Hollywood kernel. Vagrant is used for this process; a clean Ubuntu 14.04 box is downloaded, upon which the specified version of the TCP Hollywood kernel (fetched or provided in stage0) is installed.

Inputs.

- TCP Hollywood kernel
- Ubuntu 14.04 (trusty) Vagrant base box

Outputs.

- TCP Hollywood kernel Vagrant box

Stage 2 (15 minutes)

The next stage is to install the TCP Hollywood API within the Vagrant box. The TCP Hollywood API is packaged in the source code tarball described above, but if this is not available, the correct version is fetched from the GitHub repository. This stage includes installing the modified HTTP client and server used in the experiments. In addition, mininet is installed, for use in simulating the network required by the experiments.

Inputs.

- TCP Hollywood kernel Vagrant box
- TCP Hollywood API

Outputs.

- TCP Hollywood Vagrant box

Stage 3 (20-25 minutes per experiment)

The third stage involves conducting the experiments themselves, with a virtual machine (using the TCP Hollywood Vagrant box) instantiated for each run. The experiments involve streaming Big Buck Bunny over a network, simulated using Mininet. The network

[6] http://mininet.org

conditions are specified by network profiles (listed in the Makefile as `SN_RUNS` and `VN_RUNS`); these are files that describe the network conditions (e.g., bandwidth, delay, loss rates). For each profile, the experiment is run both with TCP and TCP Hollywood, to compare the two. Finally, to produce the graphs in this paper, each run is repeated. We use 10 repetitions by default; this can be controlled using the `RUN_NUMBERS` variable in the Makefile. With two protocols (standard TCP and TCP Hollywood), 23 network profiles, and 10 repetitions, the Makefile provided will perform, by default, 460 experiments.

Each run produces a set of files: logs from both the client and server, describing the application-layer activity (e.g., what is sent or received); a tcpdump taken at the server; and QoE logs (SSIM and PSNR analysis) that result from comparing the streamed video to the reference copy.

Inputs.

- TCP Hollywood Vagrant box
- Big Buck Bunny (as MPEG-DASH chunks)
- Big Buck Bunny reference (1080p); ~40GB file
- Network profiles

Outputs. Each experiment run produces:

- Client logfile
- Server logfile
- Server tcpdump
- SSIM logfile
- PSNR logfile

Stage 4 (5 minutes)

The previous stage produces output files for each experiment, giving application-layer activity, and QoE data. In this stage, these output files are aggregated, allowing for analysis and graphing in the next stage.

Broadly, the experiments fall into two groups: those with static network conditions, that do not change throughout the stream, and those with variable network conditions, where parameters, such as bandwidth and delay, are changed every 30 seconds. The results of experiments within each group, and for each network profile, are aggregated together.

Inputs.

- Stage 3 output files
- Stage 4 Python and R scripts

Outputs. Each group (static or variable) produces (aggregated by network profile):

- Aggregated PSNR data
- Aggregated SSIM data
- Aggregated QoS data (e.g., rebuffering events)
- Aggregated bitrate data

Stage 5 (5 minutes)

In this stage, the processed results files generated by the previous stage are graphed.

- Stage 4 output files
- Stage 5 R scripts

Outputs.

- Figures 3 to 16 inclusive

Stage 6 (1 minute)

With the results of the experiments graphed, the final stage is to build the paper.

- Paper TeX and BibTeX files
- Figures (static and those generated in stage 5)

Outputs.

- Paper

Discussion

The source code that we have made available, alongside this appendix, should be sufficient for repeating the experiments conducted in this paper. However, making the paper reproducible introduces a number of challenges, and encounters various limitations. We discuss and reflect on those in this section.

Each evaluation run makes use of a simulated network, run within a virtual machine. As part of this simulation, random packet loss is introduced. As this is non-deterministic, the results generated by building the paper will be different between different builds, including the published work. Typical approaches to managing randomness for repeatability are made difficult by our use of virtual machines: where ordinarily a pseudo-random number generator could be used, with a specified seed, our evaluations make use of the *system's* random number generator. By repeating our measurements a sufficient number of times (the paper, and the code we provide, specifies 10 runs), we aim not only to bolster the statistical significance of our results, but ensure that the trends we discuss hold true when the measurements are repeated. However, where we discuss a particular run (e.g., in Section 5.5), it is inevitable that the text will not match results generated in other builds. It is not clear how non-determinism should be handled when considering reproducibility, where pseudo-random number generators cannot be used.

Our particular testing requirements (i.e., a modified Linux kernel) lend themselves to using virtual machines, and orchestrating those machines programmatically (e.g., by using Vagrant). This minimises the dependencies that need to be installed on the host machine: for example, each run uses Mininet, but this runs within the virtual machine, rather than on the host machine. However, this methodology also introduces limitations: our evaluations essentially depend on a particular Linux environment, and packaging this for reproducibility is non-trivial. While including the TCP Hollywood kernel and API code goes some way towards preventing decay in the reproducibility of the paper, some dependencies on external sources remain. For example, several software packages are installed using package managers; this relies upon the availability of the underlying repositories. There exists a trade-off between how long the paper can be usefully reproduced using the assets we provide, and the tractability of identifying and including *all* of the dependencies that exist. Understanding this trade-off and making appropriate choices is important if reproducibility is to be improved more generally.

REFERENCES

[1] Andrzej Beben, P Wiśniewski, J Mongay Batalla, and Piotr Krawiec. 2016. ABMA+: lightweight and efficient algorithm for HTTP adaptive streaming. In *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2.

[2] D. Bhat, A. Rizk, and M. Zink. 2017. Not so QUIC: A Performance Study of DASH over QUIC. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. ACM, Taipei, Taiwan, 13–18. https://doi.org/10.1145/3083165.3083175

[3] Divyashri Bhat, Amr Rizk, Michael Zink, and Ralf Steinmetz. 2017. Network Assisted Content Distribution for Adaptive Bitrate Video Streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 62–75.

[4] Nassima Bouzakaria, Cyril Concolato, and Jean Le Feuvre. 2014. Overhead and performance of low latency live streaming using MPEG-DASH. In *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*. IEEE, 92–97.

[5] Alexander Frömmgen, Amr Rizk, Tobias Erbshäußer, Max Weller, Boris Koldehofe, Alejandro Buchmann, and Ralf Steinmetz. 2017. A programming model for application-defined multipath TCP scheduling. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM, 134–146.

[6] Tobias Hoßfeld, Michael Seufert, Matthias Hirth, Thomas Zinner, Phuoc Tran-Gia, and Raimund Schatz. 2011. Quantification of YouTube QoE via crowdsourcing. In *Multimedia (ISM), 2011 IEEE International Symposium on*. IEEE, 494–499.

[7] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[8] J. Iyengar and M. Thomson. 2017. QUIC: A UDP-Based Multiplexed and Secure Transport. Work in progress. (Dec. 2017).

[9] Theodoros Karagkioules, Cyril Concolato, Dimitrios Tsilimantos, and Stefan Valentin. 2017. A comparative case study of HTTP adaptive streaming algorithms in mobile networks. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 1–6.

[10] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.

[11] Stephen McQuistin, Colin Perkins, and Marwan Fayed. 2016. TCP goes to Hollywood. In *Proceedings of the 26th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 5.

[12] Stephen McQuistin, Colin Perkins, and Marwan Fayed. 2016. TCP Hollywood: An unordered, time-lined, TCP for networked multimedia applications. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*. IEEE, 422–430.

[13] Christopher Müller, Stefan Lederer, and Christian Timmerer. 2012. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *Proceedings of the 4th Workshop on Mobile Video*. ACM, 37–42.

[14] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: near-optimal bitrate adaptation for online videos. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 1–9.

[15] Emmanuel Thomas, MO van Deventer, Thomas Stockhammer, Ali C Begen, and Jeroen Famaey. 2015. Enhancing MPEG DASH performance via server and network assistance. (2015).

[16] Sheng Wei and Viswanathan Swaminathan. 2014. Low latency live video streaming over HTTP 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. ACM, 37.

[17] Mengbai Xiao, Viswanathan Swaminathan, Sheng Wei, and Songqing Chen. 2016. Dash2m: Exploring http/2 for internet streaming to mobile devices. In *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 22–31.